Terms used similar database statement

Found **37,702** of **134,837**

Sort results by relevance

Display results expanded form

Save results to a Binder

Search Tips

Open results in a new window

Try an Advanced Search

Try this search in The ACM Guide

Results 1 - 20 of 200

Result page: **1** 2 3 4 5 6 7 8 9 10 next

Best 200 shown

Relevance scale

1 Query languages: CQLF---a query language for CODASYL-type databases

Frank Manola, Alain Pirotte

June 1982 **Proceedings of the 1982 ACM SIGMOD international conference on Management of data**

Full text available: pdf(963.77 KB) Additional Information: full citation, abstract, references, citings

This paper describes CQLF (CODASYL Query Language, Flat) [MAN081]. CQLF is a high level language for accessing and manipulating data in databases described using the 1981 ANSI dpANS version of the CODASYL Data Description Language [ANSI81]. CQLF has similarities to typical relational languages, such as SQL [ASTR76, CHAM76] and QUEL [STON76]. CQLF provides capabilities for querying and operating on databases described both in a "relational style" (having no CODASYL sets, using only values to repr ...

2 Logical foundations of object-oriented and frame-based languages

Michael Kifer, Georg Lausen, James Wu

July 1995 **Journal of the ACM (JACM)**, Volume 42 Issue 4

Full text available: pdf(7.52 MB) Additional Information: full citation, abstract, references, citings, index terms, review

We propose a novel formalism, called Frame Logic (abbr., F-logic), that accounts in a clean and declarative fashion for most of the structural aspects of object-oriented and frame-based languages. These features include object identity, complex objects, inheritance, polymorphic types, query methods, encapsulation, and others. In a sense, F-logic stands in the same relationship to the object-oriented paradigm as classical predicate calculus stands to relational programming. ...

Keywords: deductive databases, frame-based languages, logic programming, nonmonotonic inheritance, object-oriented programming, proof theory, semantics, typing

3 Logical, internal, and physical reference behavior in CODASYL database systems

Wolfgang Effelsberg, Mary E. S. Loomis

June 1984 **ACM Transactions on Database Systems (TODS)**, Volume 9 Issue 2

Full text available: pdf(1.77 MB) Additional Information: full citation, abstract, references, citings, index terms, review

This work investigates one aspect of the performance of CODASYL database systems: the data reference behavior. We introduce a model of database traversals at three levels: the

logical, internal, and physical levels. The mapping between the logical and internal levels is defined by the internal schema, whereas the mapping between the internal and the physical levels depends on cluster properties of the database. Our model explains the physical reference behavior for a given sequence of DML s ...

4 Documentation production from a formal database
Christopher Hartsough, Yuzo Yamamoto, E. David Callender
January 1982 **Proceedings of the 1st annual international conference on Systems documentation**

Full text available: pdf(899.18 KB) Additional Information: full citation, abstract, references, citings, index terms

This paper reports on an existing, operational prototype system, TG/TF2, for the generation of typeset quality documentation from a formal database. TG/TF2 directly supports the conceptual separation of system design, document content design, and document format design. Specifically, support for system design is supplied by Problem Statement Language/Problem Statement Analyzer (PSL/PSA), a development of the ISDOS Project at the University of Michigan. Document content design support is pro ...

5 A retrieval technique for virtual reality databases
Venkat N. Gudivada, Jay Bhuyan, Ramesh Adusumilli
April 1997 **Proceedings of the 1997 ACM symposium on Applied computing**

Full text available: pdf(574.93 KB) Additional Information: full citation, references, index terms

Keywords: content-based image retrieval, spatial similarity, virtual reality databases

6 Interoperability of multiple autonomous databases
Witold Litwin, Leo Mark, Nick Roussopoulos
September 1990 **ACM Computing Surveys (CSUR)**, Volume 22 Issue 3

Full text available: pdf(2.66 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Database systems were a solution to the problem of shared access to heterogeneous files created by multiple autonomous applications in a centralized environment. To make data usage easier, the files were replaced by a globally integrated database. To a large extent, the idea was successful, and many databases are now accessible through local and long-haul networks. Unavoidably, users now need shared access to multiple autonomous databases. The question is what the corresponding methodology ...

7 Physical database design for relational databases
S. Finkelstein, M. Schkolnick, P. Tiberio
March 1988 **ACM Transactions on Database Systems (TODS)**, Volume 13 Issue 1

Full text available: pdf(2.99 MB) Additional Information: full citation, abstract, references, citings, index terms, review

This paper describes the concepts used in the implementation of DBDSGN, an experimental physical design tool for relational databases developed at the IBM San Jose Research Laboratory. Given a workload for System R (consisting of a set of SQL statements and their execution frequencies), DBDSGN suggests physical configurations for efficient performance. Each configuration consists of a set of indices and an ordering for each table. Workload statements are evaluated only for atomic configurat ...

8 The temporal query language TQuel

Richard Snodgrass
June 1987 **ACM Transactions on Database Systems (TODS)**, Volume 12 Issue 2

Full text available: pdf(4.12 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Recently, attention has been focused on temporal databases, representing an enterprise over time. We have developed a new language, Tquel, to query a temporal database. TQuel was designed to be a minimal extension, both syntactically and semantically, of Quel, the query language in the Ingres relational database management system. This paper discusses the language informally, then provides a tuple relational calculus semantics for the TQuel statements that ...

9 ODE (Object Database and Environment): the language and the data model
R. Agrawal, N. H. Gehani
June 1989 **ACM SIGMOD Record , Proceedings of the 1989 ACM SIGMOD international conference on Management of data**, Volume 18 Issue 2

Full text available: pdf(1.26 MB) Additional Information: full citation, abstract, references, citings, index terms

ODE is a database system and environment based on the object paradigm. It offers one integrated data model for both database and general purpose manipulation. The database is defined, queried and manipulated in the database programming language O++ which is based on C++. O++ borrows and extends the object definition facility of C++, called the class. Classes support data encapsulation and multiple inheritance. We provide facilities for creating persistent and versioned objects, defining set ...

10 Locking performance in centralized databases
Y. C. Tay, Nathan Goodman, R. Suri
December 1985 **ACM Transactions on Database Systems (TODS)**, Volume 10 Issue 4

Full text available: pdf(3.25 MB) Additional Information: full citation, abstract, references, citings, index terms

An analytic model is used to study the performance of dynamic locking. The analysis uses only the steady-state average values of the variables. The solution to the model is given by a cubic, which has exactly one valid root for the range of parametric values that is of interest. The model's predictions agree well with simulation results for transactions that require up to twenty locks. The model separates data contention from resource contention, thus facilitating an analysis of their separ ...

11 Tractable query languages for complex object databases
Stéphane Grumbach, Victor Vianu
April 1991 **Proceedings of the tenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems**

Full text available: pdf(1.24 MB) Additional Information: full citation, references, citings, index terms

12 On the family of generalized dependency constraints
John Grant, Barry E. Jacobs
October 1982 **Journal of the ACM (JACM)**, Volume 29 Issue 4

Full text available: pdf(677.44 KB) Additional Information: full citation, references, citings, index terms

13 Temporal statement modifiers
Michael H. Böhlen, Christian S. Jensen, Richard T. Snodgrass
December 2000

ACM Transactions on Database Systems (TODS), Volume 25 Issue 4

Full text available: pdf(317.23 KB) Additional Information: full citation, abstract, references, citings, index terms

A wide range of database applications manage time-varying data. Many temporal query languages have been proposed, each one the result of many carefully made yet subtly interacting design decisions. In this article we advocate a different approach to articulating a set of requirements, or desiderata, that directly imply the syntactic structure and core semantics of a temporal extension of an (arbitrary) nontemporal query language. These desiderata facilitate transitioning applications from a ...

Keywords: ATSQL, statement modifiers, temporal databases

14 The multilevel relational (MLR) data model
Ravi Sandhu, Fang Chen
November 1998 **ACM Transactions on Information and System Security (TISSEC)**, Volume 1 Issue 1

Full text available: pdf(306.07 KB) Additional Information: full citation, abstract, references, citings, index terms, review

Many multilevel relational models have been proposed; different models offer different advantages. In this paper, we adapt and refine several of the best ideas from previous models and add new ones to build the new Multilevel Relational (MLR) data model. MLR provides multilevel relations with element-level labeling as a natural extension of the traditional relational data model. MLR introduces several new concepts (notably, data-borrow integrity and the UPLEVEL statement) and significantly ...

Keywords: access control, conidentiality, multilevel security, polyinstantiation

15 ODM: an object oriented data model for design databases
M. A. Ketabchi, V. Berzins, S. March
February 1986 **Proceedings of the 1986 ACM fourteenth annual conference on Computer science**

Full text available: pdf(892.26 KB) Additional Information: full citation, references, citings

16 A structured approach for the definition of the semantics of active databases
Piero Fraternali, Letizia Tanca
December 1995 **ACM Transactions on Database Systems (TODS)**, Volume 20 Issue 4

Full text available: pdf(4.15 MB) Additional Information: full citation, abstract, references, citings, index terms, review

Active DBMSs couple database technology with rule-based programming to achieve the capability of reaction to database (and possibly external) stimuli, called events. The reactive capabilities of active databases are useful for a wide spectrum of applications, including security, view materialization, integrity checking and enforcement, or heterogeneous database integration, which makes this technology very promising for the near future. An active database system consists of ...

Keywords: active database systems, database rule processing, events, fixpoint semantics, rules, semantics

17 User interfaces: The data management facilities of PLAIN

Anthony I. Wasserman
May 1979 **Proceedings of the 1979 ACM SIGMOD international conference on Management of data**
Full text available: pdf(1.53 MB) Additional Information: full citation, abstract, references, citings

The programming language PLAIN has been designed to support the construction of interactive information systems within the framework of a systematic programming methodology. One of the key goals of PLAIN has been to achieve an effective *integration* of programming language and database management concepts, rather than either the functional interface to database operations or the low-level database navigation operations present in other schemes. PLAIN incorporates a relational database defi ...

Keywords: abstract data types, database management, information systems, interactive programs, programming language design, relational algebra, relations, type checking

18 An extension of the relational data model to incorporate ordered domains
September 2001 **ACM Transactions on Database Systems (TODS)**, Volume 26 Issue 3
Full text available: pdf(446.05 KB) Additional Information: full citation, abstract, references, index terms, review

We extend the relational data model to incorporate partial orderings into data domains, which we call the ordered relational model. Within the extended model, we define the partially ordered relational algebra (the PORA) by allowing the ordering predicate ⊑ to be used in formulae of the selection operator (σ). The PORA expresses exactly the set of all possible relations that are invariant under order-preserving automorphism of databases. This result characterizes the expressiveness ...

Keywords: Axiom system, chase rules, implication problem, language expressiveness, lexicographical ordering, mixed ordering, nonuniform completeness, order-preserving database automorphism, ordered SQL, ordered functional dependencies, ordered relational model, ordered relations, partially ordered domains, partially ordered relational algebra, pointwise ordering, tableaux, valuation mapping

19 Concepts and effectiveness of the cover-coefficient-based clustering methodology for text databases
Fazli Can, Esen A. Ozkarahan
December 1990 **ACM Transactions on Database Systems (TODS)**, Volume 15 Issue 4
Full text available: pdf(2.74 MB) Additional Information: full citation, abstract, references, citings, index terms, review

A new algorithm for document clustering is introduced. The base concept of the algorithm, the cover coefficient (CC) concept, provides a means of estimating the number of clusters within a document database and related indexing and clustering analytically. The CC concept is used also to identify the cluster seeds and to form clusters with these seeds. It is shown that the complexity of the clustering process is very low. The retrieval experiments show that the information-retrieval effectiv ...

Keywords: cluster validity, clustering-indexing relationships, cover coefficient, decoupling coefficient, document retrieval, retrieval effectiveness

20 The design of a relational database system with abstract data types for domains
Sylvia L. Osborn, T. E. Heaven
August 1986 **ACM Transactions on Database Systems (TODS)**, Volume 11 Issue 3

Full text available: pdf(913.80 KB) Additional Information: full citation, abstract, references, citings, index terms, review

An extension to the relational model is described in which domains can he arbitrarily defined as abstract data types. Operations on these data types include primitive operations, aggregates, and transformations. It is shown that these operations make the query language complete in the sense of Chandra and Harel. The system has been designed in such a way that new data types and their operations can be defined with a minimal amount of interaction with the database management system.

Results 1 - 20 of 200 Result page: **1** 2 3 4 5 6 7 8 9 10 next

Refine Search

Search Results -

Term	Documents
REUSE	97471
REUSES	2500
(11 AND REUSE).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	1
(L11 AND REUSE).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	1

3

717/141, 139

Database:

US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L12

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Monday, May 31, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L12	l11 and reuse	1	L12
L11	L10 and (match$ or similar$)	13	L11
L10	L9 and (database near statement)	14	L10
L9	(database near cursor$) and l6	73	L9
L8	L6 and (similar near database near statement)	1	L8
L7	L6 and ((find$ or search$) near (similar near statement))	0	L7
L6	707/$.ccls.	20635	L6
	DB=USPT; PLUR=YES; OP=OR		
L5	L4 and reus$	8	L5
L4	L3 and 707/$.ccls.	40	L4

L3	similar near statement	511	L3
L2	(search$ or retriev$) near (similar near statement)	0	L2
L1	(search$ or retriev$) near (similar near statement) same database	0	L1

END OF SEARCH HISTORY

First Hit

Generate Collection Print

L2: Entry 1 of 3 File: PGPB Dec 25, 2003

DOCUMENT-IDENTIFIER: US 20030236780 A1
TITLE: Method and system for implementing dynamic cache of database cursors

Summary of Invention Paragraph:
[0005] A significant level of overhead and expense is normally required to create a cursor, particularly due to the sheer amount of work involved to parse a SQL statement and to generate an execution plan for that statement. FIG. 1 is a flowchart illustrating some of the basic compilation steps that may be performed to create a cursor. In this shown approach, the SQL compilation process begins with a parse phase 150. The parse phase is so named because the SQL statement is analyzed and parsed (clause by clause) into its constituent components to create an expression tree (or parse tree). The parse phase involves syntactical analysis where the statement is analyzed for correct syntax (105) followed by a verification action, where among other things, a determination is made whether the referenced objects exist (110). User permissions may be analyzed to determine whether the requester holds sufficient access privileges to the specific objects referenced in the SQL text (115). The parse phase concludes by generating an expression tree (120). Type checking may be performed to engage data type resolution between a client process and a server process, which verifies and corrects data type incompatibilities, for example, in a heterogeneous enterprise network/system (125). An optimizer accepts the parsed and analyzed statement from the parse phase to determine an appropriate method to best execute the particular SQL statement based upon a number of criteria, including gathered statistical information, optimization methodologies, and/or selectivity analysis (130). The result is a memory resident data structure that dictates an execution plan for carrying out the database statement request. A cursor is an example of a memory structure that includes a handle to a memory location where the details and results of a parsed and optimizes statement resides. A cursor in this context is distinguishable from the use of the term cursor from other data processing contexts, such as client-side procedural language originated cursors used to aid data table processing or as a pointer to a screen location.

Summary of Invention Paragraph:
[0006] Due to this expense, it is often inefficient to recreate a cursor each and every time a client requests execution of a particular SQL statement. Instead, the cursor can be cached and reused to execute the same SQL statement. Even if a first SQL statement is not exactly the same as a second SQL statement, techniques have been developed to identify when cursors can be shared between statements that are sufficiently similar. Once example approach for sharing cursors and execution plans for database statements is described in co-pending U.S. application Ser. No. 10/086,277, filed on Feb. 28, 2002, entitled "Systems and Methods for Sharing of Execution Plans for Similar Database Statements", which is hereby incorporated by reference in its entirety.

Detail Description Paragraph:
[0021] One circumstance in which a cached cursor can be re-used is if the new SQL statement is identical to the prior query statement used to compile and create the cached cursor. In this approach, the module 212 performs a matching/comparing action to determine whether the text string of the new SQL statement exactly matches the text string of the earlier SQL statement. Even if the new SQL statement

and the earlier SQL statement are not identical, the cursor created for the earlier SQL statement can still be used to process the new SQL statement if the two statements are similar enough. For example, if the two statements only differ by only one or more literals, then the cursor can be created for a SQL statements having a "placeholder" (e.g., a bind variable) at the location of the literal, and the cursor re-used for any incarnation of that SQL statement having different literal values. One approach for implementing this type of cursor sharing is described in co-pending U.S. application Ser. No. 10/086,277, filed on Feb. 28, 2002, entitled "Systems and Methods for Sharing of Execution Plans for Similar Database Statements", which is hereby incorporated by reference in its entirety.

Detail Description Paragraph:
[0040] In this type of system architecture, a query from client 602 may cause a "thin" cursor to be created and/or cached at middle tier server 604. Instead of including all of the contents of a full cursor such as the execution plan, result set, and parse information, the thin cursor at the middle tier server 604 may contain far less information, such as the cursor identifier, the SQL statement, statement handle data structures, and references to bind and define the handle. The corresponding cursor that is more robust and resource-intensive may be created and cached at database server 614. A cache 608 may exist to hold cursor or cursor resources at middle tier server 604. A cache 616 may exist to hold cursor or cursor resources at database server 614. In this situation, the cursors 620a and 620b at the database server 614 may be created and maintained by necessity based upon opening cursors 610a and 610b at middle tier server 604.

L9: Entry 11 of 13 File: USPT Feb 23, 1999

DOCUMENT-IDENTIFIER: US 5875334 A
TITLE: System, method, and program for extending a SQL compiler for handling control statements packaged with SQL query statements

Brief Summary Text (4):
This invention relates to systems, methods, and computer programs in the field of database information processing, and more specifically to database management systems including compilers for compiling program code including query statements packaged with control or procedural statements.

Brief Summary Text (6):
The Structured Query Language (SQL) has been widely used in many relational database systems as a database language to define and manipulate databases. Some of these database systems include IBM DB2/CS, Oracle 7, Sybase Server System 10, and Informix Online Dynamic Server. (See, DATABASE 2 AIX/6000 and DATABASE 2 OS/2 SQL Reference, IBM Corporation, First Edition, October 1993; Database 2 Programming Functional Specification, IBM Corporation, Version 2.1. P481-0021, August 1994; Oracle 7 Server Application Developer's Guide, Oracle Corporation, December 1992, Part: 6695-70-0212; TRANSAC-SQL User Guide, Sybase, 1993; The INFORMIX Guide to SQL: Syntax, V.6.0, Informix, March 1994, Part 000-7597; Jim Melton, editor, (ISO-ANSI Working Draft) Database Language (SQL2), International Organization for Standardization and American National Standards Institute, 1992.)

Brief Summary Text (7):
In such systems, access of data from an application program is achieved by means of embedded SQL statements. SQL can be viewed as consisting of two main components: Data Definition Language (DDL) that defines a database, and Data Manipulation Language (DML) that manipulates and accesses data in a database. With embedded SQL statements, application programs can insert a new row into a table, delete rows from a table, update rows from a table, and selectively retrieve rows from a table. The current SQL language standard is known informally as SQL/92 or SQL2. The SQL statements embedded in an application program are translated by the host language pre-processor into invocations to certain APIs. However, SQL2 does not provide any mechanism for expressing program logic and control flow. Instead, the control flow of the application logic is written in the host language. Thus, SQL has been mainly used exclusively for database related services, and has been mostly used as an embedded language in application programs (written in C, COBOL, FORTRAN, etc.) where the host language provides the needed control logic of an application.

Brief Summary Text (8):
Since each SQL statement is compiled into an API call to database server routines executed by the server, and the result is sent back to the application program, frequent communication and data transfer between the client and the server results in low performance. The current trend is to push more logic into the server side, so that the client application can get back only the data it really needs.

Brief Summary Text (9):
The upcoming SQL3 standard (See, Jim Melton, editor, (ISO-ANSI Working Draft) Database Language (SQL3), International Organization for Standardization and American National Standards Institute, August 1994) has defined a procedural

extension to the existing SQL2 language. The SQL3 standard has included a new set of statements, called control statements (also referred to herein as procedural statements or procedural constructs), that allow control logic to be incorporated into an SQL statement. The purpose is to make SQL a more computationally-complete language so that computations which have no direct impact on the client side can be moved to the server side to reduce the amount of data transferred between the server and the application. As a consequence, significant performance improvement can be achieved. With SQL3, control logic that used to be expressed only in a host language can now be packaged with query statements to be serviced by the server in a single dialog. The inclusion of these new control statements greatly enhances the expressiveness of the language and enables various programming techniques that used to rely on host languages.

Brief Summary Text (11):
There are four different program contexts where these SQL3 control statements can be used to improve performance of the application program and the functionality provided by the database engine: (1) as an embedded segment of the host program; (2) as the function body of a user-defined function (DB2/CS V2 SQL Referenced); (3) as the body of a stored procedure, and (4) as the body of a trigger (See, Database 2 Programming Functional Specification, IBM Corporation, Version 2.1. P481-0021, August 1994). These four program contexts are discussed in more detail as follows.

Brief Summary Text (14):
In the example shown above, a compound statement, BEGIN . . . END, is embedded in the host program, following the code which acquires the current trading information from the trading market. This compound statement inserts the current trading information into the tradings table and, depending on the trading.sub.-- type, it updates the selling.sub.-- price column or the buying.sub.-- price column of the corresponding row of the stock table. Since the compound statement is executed entirely on the server side, all the query statements (SELECT, INSERT, and UPDATE) it contains will be serviced in a single communication between the client program and the database server. Such a database access scenario could have taken three communications to complete if it were programmed without using the compound statement.

Brief Summary Text (15):
As demonstrated by this example, there are three advantages provided by the SQL3 control statements. First, in general, the number of communications between the client and the server are reduced by packaging related database access operations with control statements. Therefore, performance of the application program is significantly improved, especially in the environment where communication cost is significant (e.g. multi-media applications with a remote database server). Second, unnecessary bind-in and bind-out is avoided if the output of one query statement is used only as the input to another query statement. Third, readability of the application program is also improved because related database access operations can be packaged into one single SQL statement.

Brief Summary Text (17):
Control statements are also useful in composing the function body of a user-defined function. For example, the database access session as shown in the previous subsection can be modified and packaged as a user-defined function.

Brief Summary Text (24):
Now the database will maintain the daily high trading record and the daily low trading record up-to-date whenever a new record is inserted into the tradings table.

Brief Summary Text (25):
As shown by the examples and discussion above, there are several advantages to the procedural extensions of SQL3. The improved performance gains by means of these

procedural constructs is highly desired by the application developers. The performance is enhanced by minimizing network communication traffic between a client application and server host. The procedural constructs allow "control abstraction" by grouping SQL statements together procedurally resulting in fewer bind in/bind out processes. Instead of the server handling one SQL statement at a time as for SQL2, SQL3 enables the server to perform global optimization of the query statements. Also, because of SQL3's procedural aspects, SQL3 is now easily adaptable to object extensions since objects also contain methods or procedures. A procedural extension to SQL is a prerequisite to handling objects

Brief Summary Text (28):
In an SQL-based environment, an SQL statement interacts with the host program by means of a set of host variables. Host variables are made known to the SQL compiler at the early stage of the application preparation-time. A reference to a host variable of certain type in the SQL statement is compiled into a plan object (run-time object) of the corresponding SQL type. Host variables could be used for the application to provide data to the database server (input host variable) or to receive data from the database server (output host variable), depending on the context where the host variable appears in the SQL statement. The same host variable can have multiple occurrences in an SQL statement, as an input host variable, an output host variable, or both. At run-time, all the input host variables are packed into a variable-length structure, input SQLDA, that is transferred to the database engine as input parameters to the execution plan of the corresponding SQL statement. The very first action taking place in the execution plan is to convert the data contained in the input SQLDA to the corresponding plan objects which are directly accessed by the execution plan. Such a process is usually referred to as bind-in process. Similarly, for output host variables, there is a bind-out process taking place at the end of the execution plan to convert the plan objects to their corresponding SQLDA entries.

Brief Summary Text (29):
In SQL2, the mapping from SQLDA entries to the corresponding plan objects is simple. For example, the n-th SQLDA entry corresponds to the n-th host variable occurrence in the SQL statement. However, such a simple mapping no longer works for a SQL3 control statement because there can be more than one SQL statement embedded in a control statement. As a result, there can be multiple orders of host variables such that the order of host variable occurrences in one statement does not necessarily agree with that of the statements. Also, there may be sharing of host variables among the statements such that one host variable may be an output variable in one statement, but an input variable in a subsequent statement. This problem did not occur in SQL2 since the server was only handling one SQL statement at a time.

Brief Summary Text (31):
Local variables are new SQL objects introduced in the SQL3 standard. Unlike column objects, the same local variable is accessible in multiple SQL statements. Besides, local variables are associated with a scope where they are defined. Therefore, a local variable of an inner scope will make another local variable with the same name in an outer scope invisible from the inner scope. Similarly to host variables having multiple orders, local variables can have nested scopes. Also similar to the problem with host variables, local variables can be shared among the multiple statements. How to handle local variables at run-time as well as compile-time becomes a new issue to the existing SQL2 compiler.

Brief Summary Paragraph Table (1):
____________________________________ main( ) . . . /* Get trading information: performed by the host program. */ . . . /* Update database: performed by the control statement. */ EXEC SQL BEGIN DECLARE company.sub.-- id INTEGER; /* Look up the unique ID by name. */ SELECT id INTO company.sub.-- id FROM company WHERE strcmp(name, :company.sub.-- name) = 0; /* Insert current trading information. */

INSERT INTO tradings VALUES (id, :trading.sub.-- type, :trading.sub.-- time, :unit.sub.-- price, :total.sub.-- shares); /* Update the selling or buying price. */ IF (:trading.sub.-- type = 0) THEN UPDATE stock SET selling.sub.-- price = :unit.sub.-- price WHERE id = company.sub.-- id; ELSEIF (:trading.sub.-- type = 1) THEN UPDATE stock SET buying price = :unit.sub.-- price WHERE id = company.sub.-- id; END IF; END; /* Host program continues here. */ . . . }

Detailed Description Text (2):
In DB2/CS V2 query compiler, most SQL statements are parsed into an intermediate representation called Query Graph Model (QGM). A QGM unit is a directed graph (although it could be cyclic due to recursive queries) in which each node represents a table transformation function and each edge flows the rows of the table produced by a predecessor table function into its successor table function. A QGM node without incoming edge represents a constant table function where the output is a constant table (e.g., base table). Once created by the parser, a QGM unit is used as a representation of the original SQL statement in subsequent compiler phases. QGM is a powerful mechanism for representing SQL statements.

Detailed Description Text (7):
In SQL3 standard, SQL statements are divided into seven different classes as shown in the following syntax rule.

Detailed Description Text (22):
The REPEAT statement is similar to the WHILE statement except that the SQL <search condition> is tested at the end of each iteration. Therefore, the pertaining statements will be executed at least once.

Detailed Description Text (32):
The FOR statement executes a sequence of statements for each row of the table specified by the <cursor specification>. Columns of the table can be accessed by the pertaining statements as if they were local variables whose value is bound to that of the corresponding column.

Detailed Description Text (35):
Database Language Statement herein refers to meaning one or more statements grouped together including but not limited to SQL procedural, control, declarative, query, or other kinds of SQL statements as well as other database language statements.

Detailed Description Text (37):
SQLDA shall mean SQL descriptor area. A structure used to pass data to and receive data from the database engine.

Detailed Description Text (38):
Bind-In/Bind-Out shall mean the process whereby the host variables of an application's embedded SQL statements are prepared for runtime access and execution on a database.

Detailed Description Text (40):
RDS shall mean Relational Data System. A component in a Relational database system. This component includes the parser, optimizer and code generation.

Detailed Description Text (46):
Most SQL3 control statements are "procedural" statements in that they define an evaluation order for the execution of the program. Unlike other SQL statements, a control statement does not operate on tables. For example, a compound statement defines a new scope and groups a sequence of statements together and a WHILE statement repeats a sequence of statements based on the value of an SQL search condition.

Detailed Description Text (54):
FIG. 2 illustrates such transformations by extracting SQL query statements 102 and producing a SAP skeleton 103. The connections are shown between the original input SQL statement, the SQL.sub.-- STUBs in SAP skeleton and the extracted query statements. The input control statement 101 is shown at the top. The result of pre-processing is the SAP skeleton 103 shown at the bottom left and a set of extracted SQL query statements 102 shown at the bottom right. Each SET statement 211 has been extracted and converted into a VALUES INTO statement 221, and a SQL.sub.-- STUB 231 is inserted in the SAP skeleton. We also notice that the search condition a>b of the IF statement 212 is extracted and is replaced by a compiler-generated variable, #SAPtmp#, whose value is set by another VALUES INTO statement 222 indicated by the third query statement.

Detailed Description Text (56):
SQL statements: these are the statements that can be processed by the current compiler to generate access plans. After this phase, they will be replaced by SQL.sub.-- STUB.

Detailed Description Text (120):
The following discusses the Query Compiler path and compiling query statements. The previous section describes how the SAP skeleton (i.e., sap.sub.-- cmpl.sub.-- unit [0]) is processed. In this section we describe how the other compilation units sap.sub.-- cmpl.sub.-- unit[i], i>0, are processed. Recall that these compilation units are extracted because they can be compiled by the existing query compiler path without any change, almost. Each of these compilation units can be viewed just as an independent input SQL statement to the query compiler to obtain their access plans. It is the responsibility of SAP skeleton code generation to glue these plans together.

Detailed Description Text (124):
The following describes some of the issues in this phase. The term "an input SQL statement" will be used to mean one of these compilation units.

Detailed Description Text (126):
Recall that each compilation unit has a field scope that records the context where this input SQL statement is in. Each scope 804, 805 defines a symbol table 802, 803 and has a link to its parent scope (as shown in FIG. 8). If the input SQL statement is embedded in a compound statement, then it may reference some of the local variables defined in any of its enclosing compound statements. Thus, a symbol table look-up may go up to the symbol table in the parent scope or further if a variable is not in the current scope.

Detailed Description Text (128):
As mentioned before, local variables are treated as if they were host variables to minimize the impact to the current compiler. For each occurrence of a host variable in an SQL statement, there is a plan object allocated by the code generator in the access plan, and the bind-in/bind-out process simply maps the n-th plan object to the n-th SQLDA entry. The following describes the work to be done before code generation in order to accomplish this for local variables, so that bind-in/bind-out can be correctly handled.

Detailed Description Text (132):
The SELECT statement will be extracted into a separate compilation unit. Since only x is known by the pre-compiler as a host variable, there is only one SQLDA entry 704 created. A simulated, i.e.,fake, SQLDA 162 is created, where two entries 705, 706 are inserted for local variables a and b. This simulated SQLDA is used for the bind-in/bind-out operation of this extracted compilation unit. The term local SQLDA 162 will be used to refer to this simulated SQLDA because this SQLDA is local to each extracted SQL statement. Since local variables a and b are treated as host variables in QGM (possessing column objects), they, like x, will have plan objects

701 allocated in the plan. See FIG. 7B. Another global storage 703 is allocated in the meta-plan of the SAP skeleton, which acts as a "persistent" store that allows the values of local variables to live across multiple SQL statements.

Detailed Description Text (133):
In constructing the local SQLDA for an extracted SQL statement, like the way host variables are treated currently, each occurrence of host variables, as well as local variables, will be numbered (as before, input and output are numbered separately). For example,

Detailed Description Text (134):
where the superscript ik(ok) denotes input (output) variable number k. In addition, the two maps, var.sub.-- ref.sub.-- map and var.sub.-- upd.sub.-- map, map a local variable to its symbol table entry and map a real host variable to its corresponding SQLDA entry that is supplied by the pre-compiler. The number given to a variable will be used as an index to one of the two maps. During the meta-plan generation, a plan object will be allocated for each local variable, and the pointer to the object will be stored in the corresponding symbol table entry. Therefore, during the plan generation of this SQL statement, a local SQLDA can be constructed according to the information stored in the two maps; that is, a host variable will have an entry copied from real SQLDA, while a local variable will have an entry copied from the meta-plan plan object, through its symbol table entry.

Detailed Description Text (176):
The REPEAT statement is treated in a way similar to that of the WHILE statement. However, the IF instruction is placed after the loop body. The second field of the IF instruction now points back to the beginning of the loop body. This is achieved by saving the offset of the loop body head in a local variable. The layout of the threaded code 213 is shown in FIG. 21.

Detailed Description Text (186):
A meta-plan may contain many sub-plans. Each sub-plan corresponds to an SQL statement in the control statement program. All the sub-plans are chained together in a form of a link-list. After the meta-plan is loaded, the pointer in each sub-plan needs to be relocated.

Detailed Description Text (202):
There are four kinds of data structure used in the SAP compiler. At the top-level is a SAP control block, sap.sub.-- cb, which organizes all the data structures used in the SAP compiler. The sap.sub.-- cb control block is anchored from the RDS control block. Therefore, it can be accessed by every component of the database engine.

Detailed Description Text (305):
Cursors Declared in Host Program

Detailed Description Text (306):
In the V2 compiler, declaration and reference of a cursor is processed in a way which involves the collaboration between the pre-processor and the SQL compiler. The SQL statement associated with the cursor is compiled into an execution plan by the SQL compiler. The pre-processor maintains the mapping between the cursor name and the execution plan. Reference to the cursor in an open statement or a fetch statement is translated into an invocation to some API with the plan ID passed as a parameter.

Detailed Description Text (307):
To support cursor fetch in a control statement, the RDS component can be extended with a new service request, add cursor, for the pre-processor to inform the SQL compiler of the mapping between the cursor name and the corresponding execution

plan. The add cursor request should work in the same way as the add host variable request. With such an extension to RDS, the code synthesizer will generate a special op-code for the fetch statement. At run time, the special op-code will be interpreted as an execute plan instruction and will cause the corresponding plan to be executed. At its completion, the original meta-plan will be resumed to continue the execution.

Detailed Description Text (311):
Control statements may provide the compiler with better information for query optimization. For example, because of the compound statement, multiple-query optimization (See, Timos K. Sellis, Multiple-Query Optimization, ACM Transaction on Database Systems, 13(1):23-52, March 1988) becomes possible and can be important. Let Emp be a table with five columns: name, emp.sub.-- num, age, job, and dept.sub.-- name. Let Dept be a table with columns name and num.sub.-- of.sub.-- emp. The following queries can be optimized in a way such that fewer temporary tables are generated.

Detailed Description Paragraph Table (3):
<compound statement> ::= [<beginning.sub.-- label> <colon>] BEGIN [[NOT] ATOMIC] [<local.sub.-- declaration.sub.-- list>] [<SQL statement list>] END [<ending label>] <SQL statement list> ::= <SQL procedure statement> <colon> . . . <local declaration list> ::= <local declaration> <colon> . . . <local declaration> ::= <SQL variable declaration> .vertline. . . . <SQL variable declaration> ::= DECLARE <SQL variable name list> [<constant or updatable>] .vertline. <data type> .vertline. <domain name> " [<default clause>]

Detailed Description Paragraph Table (5):
<if statement> ::= IF <search condition> THEN <SQL statement list> [ELSEIF <search condition> THEN <SQL statement list>] [ELSE <SQL statement list>] END IF

Detailed Description Paragraph Table (6):
<loop statement> ::= [<beginning label> <colon>] LOOP <SQL statement list> END LOOP { <ending label> }

Detailed Description Paragraph Table (7):
<while statement> ::= [<beginning label> <colon>] WHILE <search condition> DO <SQL statement list> END WHILE [<ending label>]

Detailed Description Paragraph Table (8):
<repeat statement> ::= [<beginning label> <colon>] REPEAT <SQL statement list> UNTIL <search condition> END REPEAT [<ending label>]

Detailed Description Paragraph Table (9):
<case statement> ::= <simple case statement> .vertline. <searched case statement> <simple case statement> ::= CASE <value expression> <simple case statement when clause> . . . [ELSE <SQL statement list>] END CASE <simple case statement when clause> ::= WHEN <value expression> THEN <SQL statement list> <searched case statement> ::= CASE <search case statement when clause> . . . [ELSE <SQL statement list>] END CASE <searched case statement when clause> ::= WHEN <search condition> THEN <SQL statement list>

Detailed Description Paragraph Table (13):
[<beginning label> <colon>] FOR <identifier> AS [<cursor name> [<cursor sensitivty>] CURSOR FOR] <cursor

specification> DO <SQL statement list> END FOR [<ending label>]

Detailed Description Paragraph Table (51):

BEGIN DECLARE c1 CURSOR FOR SELECT * FROM Emp, Dept WHERE Emp.dept.sub.-- name = Dept.name AND Emp.age <= 50 AND Dept.num.sub.-- of.sub.-- emp <= 20; DECLARE c2 CURSOR FOR SELECT * FROM Emp, Dept WHERE Emp.dept.sub.-- name = Dept.name AND Emp.age <= 40 AND Dept.num.sub.-- of.sub.-- emp <= 30; OPEN c1; OPEN c2; loop1: LOOP FETCH c1 INTO . . . ; IF (. . .) THEN LEAVE loop1; . . . END LOOP loop1; /* No insert, update, or delete on Emp and Dept. */ . . . loop2: LOOP FETCH c2 into . . . ; IF (. . .) THEN LEAVE loop2; . . . END LOOP loop2; . . . END

CLAIMS:

1. A system for compiling a database language statement having a procedural part packaged together with at least one declarative part, the system comprising:

a query extractor for separating the procedural part from the declarative part whereby the procedural part, separated from the declarative part, is a control skeleton;

an integrated compiler having a query compiler and a control analyzer wherein said control analyzer is specialized for compiling the procedural part;

the query compiler compiling the at least one declarative part and generating an executable plan;

the control analyzer generating a control flow representation of control flow information of the control skeleton; and

a plan synthesizer for synthesizing the output from the query compiler and the control analyzer by merging a code sequence generated from the control flow information of the control skeleton with the executable plan.

2. The system of claim 1 wherein the control skeleton comprises the procedural part of the database language statement and a token in place of the declarative part.

3. The system of claim 1 wherein the control flow representation preserves a semantic of the database language statement.

15. The system of claim 1 wherein the control analyzer generates at least one scope and a symbol table when a compound statement is encountered in the database language statement.

21. A system for compiling a database language statement having a procedural part packaged with at least one declarative part, the system comprising:

a parser for separating the procedural part from the declarative part whereby the procedural part, separated from the declarative part, is a control skeleton, and for generating a first representation of the declarative part and a control flow representation of the control skeleton;

an analyzer creating a table dependency graph from each declarative part;

a threaded code generator for generating a code sequence representing the control flow of the control skeleton;

a query rewrite applying rewrite rules to the declarative part based upon information from the analyzer to rewrite a query for each declarative part;

a query optimizer optimizing the code sequence and the plan for the declarative part based upon the information from the analyzer; and

a plan synthesizer synthesizing a final plan from an optimized code sequence and an optimized execution plan from the query optimizer.

23. A system for compiling a database language statement having a procedural part packaged together with at least one declarative part, the database language statement embedded in an application program for querying a database managed by a server, the system comprising:

an integrated compiler having a first compilation path for compiling each declarative part apart from the procedural part and having a second compilation path for processing the procedural part;

a control analyzer, in the second compilation path, for analyzing a control flow of said procedural part; populating a host variable table with local variables, and creating a symbol table of local and host variables;

a query compiler, in the first compilation path for handling all local variables as host variables from the host variable table populated with local variables in compiling each declarative part into an executable plan;

a threaded code generator, in the second compilation path, for generating a threaded code sequence of the control flow of the procedural part;

a plan synthesizer for embedding each executable plan into the threaded code sequence thereby creating a meta-plan, and allocating storage for local and host variables from the symbol table during code generation time;

a run time interpreter recursively executing each execution plan embedded within the threaded code sequence within the meta-plan wherein the meta-plan has one bind-in operator and one bind-out operator for communicating with the application input and output values of local and host variables.

24. A method for compiling a database language statement having a procedural part packaged together with at least one declarative part, said method comprising:

separating the procedural part from the declarative part;

generating, by a query compiler, an executable plan for each of the at least one declarative part;

generating a threaded code sequence based upon control flow information from the procedural part; and

embedding the executable plan for each of the at least one declarative part as a sub plan into the threaded code sequence to create an executable meta-plan.

31. A method for compiling a database language statement having a procedural part packaged together with at least one declarative part, the method comprising:

separating the database language statement into the procedural part and the declarative part;

utilizing a query compilation path, for the declarative part, for generating an executable plan for each one of the at least one declarative part;

utilizing a second compilation path, separate from said query compilation path, for

the procedural part, for generating control flow information and a threaded code sequence based upon the control flow information; and

synthesizing a final execution plan for the database language statement from the threaded code sequence and the executable plan for each one of the at least one declarative part.

32. The method of claim 31 wherein the step of separating the database language statement further comprises:

transforming a SET statement into an equivalent query statement thereby directly translating the procedural part to the declarative part;

extracting a search condition of a procedural part and replacing the search condition with a compiler generated local variable and, based upon the search condition, assigning a first representation for true or a second representation for false to the local variable, thereby creating a pure procedural part; and

replacing each declarative part with a token to create a control skeleton for the procedural part of the database language statement.

34. A method for compiling a database language statement having a procedural part packaged together with at least one declarative part, the method comprising:

transforming a SET statement into an equivalent query statement thereby directly translating the procedural part to a pure declarative part;

extracting a search condition of a procedural part and replacing the search condition with a compiler generated local variable and, based upon the search condition, assigning a first representation for true or a second representation for false to the local variable, thereby creating a pure procedural part;

replacing each declarative part with a token to create a control skeleton representing the procedural part of the database language statement; and

compiling each declarative part through a query compiler and processing the control skeleton separate from the declarative part.

35. A method of compiling a database language statement having a procedural part packaged together with at least one declarative part, the method comprising:

separating the procedural part and the declarative part into a control skeleton and the declarative part;

compiling, through a first compilation path through a query compiler, each declarative part into an execution plan;

generating, separately from said first compilation path, a threaded code sequence representing a control flow of the control skeleton; and

synthesizing the threaded code sequence and each execution plan into one meta-plan.

36. A system for compiling a database language statement having a procedural part packaged together with at least one declarative part, said system comprising:

means for separating the procedural part from the declarative part;

means for processing the declarative part through a query compiler to generate an executable plan for each of the at least one declarative part;

means, separate from the query compiler, for analyzing control flow information of the procedural part and generating a threaded code sequence based upon the control flow information; and

means for merging the executable plan for each of the at least one declarative part into the threaded code sequence to create an executable meta-plan.

39. A system for compiling a database language statement having a procedural part packaged together with at least one declarative part, said system comprising:

means for separating the procedural part from the declarative part;

means for integrating a compilation of the declarative part in a first compilation path and a processing of the procedural part in a second path;

means, in the first compilation path, for compiling each declarative part into an execution plan;

means, in the second path for generating a threaded code sequence representing a control flow of the procedural part; and

means for synthesizing the threaded code sequence and each execution plan into a meta-plan.

42. The system of claim 39 further comprising means for populating a table with both local and host variables during a parsing phase of the database language statement and means for allocating storage for all variables in the table during a generation of code.

46. A system for compiling a database language statement having a procedural part packaged with at least one declarative part, the system comprising:

means for separating the procedural part from the declarative part into a control skeleton and the at least one declarative part;

means for generating a first representation of the declarative part and generating a control flow representation of a control flow of the control skeleton;

means for performing an analysis on the control flow representation and creating a table dependency graph from each declarative part;

means for generating a code sequence representing the control flow of the control skeleton;

means for applying rewrite rules to the declarative part, based upon information from the means for performing an analysis, for generating a plan for each declarative part;

means for optimizing the code sequence and the plan for the declarative part based upon the information from the means for performing an analysis; and

means for synthesizing a final plan from an optimized code sequence and an optimized execution plan from the means for optimizing.

47. An article of manufacture having a computer usable medium having computer readable program code means embodied therein for enabling a compilation of a database language statement having a procedural part packaged together with at least one declarative part, said article of manufacture comprising:

means for enabling a separation of the procedural part from the declarative part;

means for enabling an integrated compilation of the declarative part in a first compilation path and a processing of the procedural part in a second path;

means for enabling, in the first compilation path, a compilation of each declarative part into an execution plan by a query compiler;

means for enabling, in the second path, a generation of a threaded code sequence representing a control flow of the procedural part; and

means for enabling a merging of the threaded code sequence and each execution plan into a meta-plan.

48. A system for compiling a database language statement having a procedural part packaged together with at least one declarative part, the system comprising:

means for compiling each declarative part into an executable plan;

means for generating a threaded code sequence of a control flow of the procedural part; and

means for embedding each executable plan into the threaded code sequence as a single executable meta-plan.

50. A method for compiling a database language statement having a procedural part packaged together with at least one declarative part, the method comprising:

compiling each declarative part into an executable plan;

generating a threaded code sequence of a control flow of the procedural part; and

embedding each executable plan into the threaded code sequence.

51. An article of manufacture having a computer usable medium having computer readable program code means embodied therein for enabling a compilation of a database language statement having a procedural part packaged together with at least one declarative part, said article of manufacture comprising:

means for enabling a compilation of each declarative part into an executable plan;

means for enabling a generation of a threaded code sequence of a control flow of the procedural part; and

means for enabling an embedding of each executable plan into the threaded code sequence as a single executable meta-plan.

Print Selection

Clear Cancel Print Print First Page

Section: Page(s): Print Copy: 1

Select?	Document ID	Section(s)	Page(s)	# Pages to print	Database
☑	5875334	all	all	43	PGPB,USPT,USOC,EPAB,JPAB,DWPI
☑	5758144	all	all	40	PGPB,USPT,USOC,EPAB,JPAB,DWPI
☑	5734884	all	all	40	PGPB,USPT,USOC,EPAB,JPAB,DWPI

Building	Room	Printer
cpk2	4c32	gbfrptr

Freeform Search

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term: search$ same (similar near statement)

Display: 50 Documents in Display Format: - Starting with Number 1

Generate: ○ Hit List ◉ Hit Count ○ Side by Side ○ Image

Search Clear Interrupt

Search History

DATE: Monday, May 31, 2004 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L19	search$ same (similar near statement)	6	L19
L18	L17 and reuse	11	L18
L17	L16 and cursor$	45	L17
L16	L15 and database	119	L16
L15	search$ and (similar near statement)	193	L15
L14	(dissimilar) near statement$	1	L14
L13	L12 and (sql near statement)	13	L13
L12	L11 and cursor$	54	L12
L11	L10 and database	191	L11
L10	(similar or dissimilar) near statement$	784	L10
L9	L8 and (sql near statement)	13	L9
L8	l7 and database	54	L8
L7	L6 and (similar near statement)	84	L7
L6	L3 and (cursor$)	84	L6
L5	L3 and (cursor$ near database)	1	L5
L4	L3 and (cursor$ near databasse)	0	L4

First Hit Fwd Refs

Generate Collection Print

L9: Entry 4 of 13 File: USPT Aug 6, 2002

DOCUMENT-IDENTIFIER: US 6430550 B1
**** See image for Certificate of Correction ****
TITLE: Parallel distinct aggregates

Brief Summary Text (4):
In a database management system (DBMS), data is stored in one or more data containers, each container contains records, and the data within each record is organized into one or more fields. In relational database systems, the data containers are referred to as tables, the records are referred to as rows, and the fields are referred to as columns. In object oriented databases, the data containers are referred to as object classes, the records are referred to as objects, and the fields are referred to as attributes. Other database architectures may use other terminology.

Brief Summary Text (5):
The present invention is not limited to any particular type of data container or database architecture. However, for the purpose of explanation, the examples and the terminology used herein shall be that typically associated with relational databases. Thus, the terms "table", "row" and "column" shall be used herein to refer respectively to the data container, record, and field.

Brief Summary Text (6):
In typical database systems, users write, update and retrieve information by submitting commands to a database application. To be correctly processed, the commands must comply with the database language that is supported by the database application. One popular database language is known as Structured Query Language (SQL).

Brief Summary Text (7):
Multi-processing systems are typically partitioned into nodes, where each node may contain multiple processors executing multiple concurrent processes. To fully utilize the computing power of a multi-processing system, a database application may divide a large processing task required by a query into smaller work granules which may then distributed to processes running on one or more processing nodes. Because the various work granules are being performed in parallel, the processing required by the query can be completed much faster than if the processing were performed on a single node by a single process. One mechanism for implementing parallel operations in a database management system is described in U.S. patent application Ser. No. 08/441,527 entitled "Method and Apparatus for Implementing Parallel Operations in a Database Management System" filed on May 15, 1995, by Gary Hallmark and Daniel Leary, incorporated herein by reference.

Brief Summary Text (11):
Query Q1 returns the number of distinct managers there are in each region. During execution of this query, the database server (1) groups together rows by region value, (2) eliminates, within each region group, rows that have duplicate manager values, and (3) counts how many rows remain in each region group after the duplicates have been removed. In table 100 illustrated in FIG. 1, this operation results in the values: 2 for region N, 2 for region S, 1 for region E, and 1 for region W.

Detailed Description Text (101):
In the foregoing description, the tasks that are assigned to the slave processes of the various row sources used in a parallel distinct aggregate operation have been described. The mechanism by which the slave processes are instructed to perform their assigned tasks may vary from implementation to implementation. The present invention is not limited to any particular technique for instructing slave processes. One technique that may be used to instruct slave processes, according to one embodiment, involves generating SQL statements that specify the specific tasks required of the slave processes, and passing the SQL statements to the appropriate slave processes for execution. In one embodiment that generates slave-process-specific SQL statements, the operators "MAKE_PSR" and "CAST_PSR" are used.

Detailed Description Text (103):
In one embodiment, the SQL statements issued to the slave processes that belong to the first-stage row source that is executing Q3 use MAKE_PSR as follows: select region, MAKE_PSR ((count (distinct mgr)), (count (distinct age))) from X group by region

Detailed Description Text (105):
The slave-process-specific SQL statements issued to the slave processes that belong to the second-stage row source may have the following form: select region, MAKE_PSR ((count (distinct (CAST_PSR(mgr, 1)))), (count (distinct (CAST_PSR(age, 2))))) from RS1 group by region

Detailed Description Text (107):
The slave-process-specific SQL statements issued to the slave processes that belong to the third-stage row source may have the following form: select region, (count (distinct (CAST_PSR(mgr, mgr-code)))), (count (distinct (CAST_PSR(age, age-code)))) from RS2 group by region

Detailed Description Text (108):
This statement is similar to those executed at the second-stage. However, MAKE_PSR is not invoked because the third stage produces result rows, where each row combines the row pieces associated with each of the distinct-key columns.

Detailed Description Text (111):
Computer system 500 may be coupled via bus 502 to a display 512, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 514, including alphanumeric and other keys, is coupled to bus 502 for communicating information and command selections to processor 504. Another type of user input device is cursor control 516, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 504 and for controlling cursor movement on display 512. This input device typically has two degrees of freedom in two axes, a second axis (e.g., x) and a third axis (e.g., y), that allows the device to specify positions in a plane.

First Hit Fwd Refs

Generate Collection Print

L5: Entry 5 of 8 File: USPT Oct 5, 1999

DOCUMENT-IDENTIFIER: US 5963742 A
TITLE: Using speculative parsing to process complex input data

Brief Summary Text (14):
An example of a statement which is similar to the statement above is:

Detailed Description Text (26):
Development time can also be much faster. Not only does the present invention allow parsers to be designed in a top-down manner (which is likely to be the way the input format is documented), but it makes it easier for multiple developers to design different parts of one tool at the same time. If the subparsers call each other, code reusability is improved as well.

Detailed Description Text (27):
A further exemplary embodiment of the present invention is illustrated in FIG. 6. In accordance with FIG. 6, a single re-entrant parser with multiple entry points is used in place of a traditional monolithic single parser. Speculative parsing is used, enabling each entry point to act as an independent parser which need not be aware of the other entry points. Each entry point need only attempt to decode its input, reporting a confidence level as well as speculative results. For each statement in the input data, multiple entry points are invoked which independently attempt to decode the respective statement. Furthermore, a production can be accessed from more than one entry point and is hence reusable.

Current US Cross Reference Classification (2):
707/1

First Hit Fwd Refs

☐ Generate Collection Print

L11: Entry 8 of 13 File: USPT Apr 13, 2004

DOCUMENT-IDENTIFIER: US 6721731 B2
TITLE: Method, system, and program for processing a fetch request for a target row at an absolute position from a first entry in a table

Brief Summary Text (3):
The present invention relates to a method, system, and program for implementing cursors in a database and, in particular, updateable scrollable cursors.

Brief Summary Text (9):
The ODBC defines the following types of cursors: scrollable cursor: allows the application to fetch forward or backward from the current position, i.e., from anywhere, in the result set. With a scrollable cursor, your application can request by position the data presented in the current row. Typical scrolling requests include moving one row forward, one row back, to the beginning, or to the end of the result set. With a scrollable cursor, the application can request that a certain row of data be made the current row more than once. forward-only cursor: allows the application to fetch forward serially from the start to end of the result set. keyset cursor: the rows in the result table are identified by the value present in a designated column. static cursors only contain data that was placed in the cursor when it was created. A static cursor does not display new rows inserted in the database after the cursor was opened, even if they match the search conditions of the cursor SELECT statement. If rows in the result table are updated by means other than through the cursor defining the result table, then the new data values are not displayed in the static cursor. The static cursor may display rows deleted from the database after the cursor was opened if they were deleted by a positioned delete through the cursor. dynamic cursors: Dynamic cursors reflect all changes made to the rows in their result table when scrolling through the cursor. The data values, order, and membership of the rows in the result table can change on each fetch. All UPDATE, INSERT, and DELETE statements made by all users are visible through the cursor. Updates are visible immediately if they are made through the cursor. Updates made outside the cursor are not visible until they are committed, unless the cursor transaction isolation level is set to read uncommitted. Updates made outside the cursor by the same transaction as that which defines the cursor are immediately visible.

Detailed Description Text (4):
The executive program 6 initially receives the database statements from the application 2. Upon detecting that the commands from the application 2 is an SQL query or cursor related command, the executive program 6 would call the parser precompiler 8 to parse the statements from the application program 2 and generate parse trees for the statements in a manner known in the art. The parser precompiler 8 would return the parsed statements to the executive 6. The executive program 6 would then call the optimizer program 10 to optimize the parse trees in a manner known in the art and return the optimized parse trees to the executive 6. The executive 6 would then call the structure generator 12 program to generate the control blocks and related data structures to implement the cursors in accordance with the preferred embodiments. The structure generator 12 receives as input the optimized parsed trees of the statements to build the runtime structure.

Detailed Description Text (65):

With respect to FIG. 13, a database page set control record 450 for a table space includes an open scrollable cursor table 460 and a RID table 480. Each entry in the open scrollable cursor table 460 includes a scrollable cursor identifier field 462 that uniquely identifies the scrollable cursor in the database and a time stamp field 464, which in the described embodiments comprises a log record sequence number (LRSN) that indicates a time value at which the scrollable cursor having the scrollable cursor ID was opened. The open scrollable cursor table 462 identifies open cursors that include result table 50 entries having base table entries included in the table space for which the database page set control record 450 is maintained. Thus, a single open scrollable cursor can have entries in the open scrollable cursor table 462 in multiple database page set control records 450, for each table space having base table entries maintained in the scrollable cursor.

Current US Original Classification (1):
707/3

Current US Cross Reference Classification (1):
707/4

Print Request Result(s)

Printer Name: cpk2_4c32_gbfrptr
Printer Location: cpk2__4c32
Number of Copies Printed : 1

- US006721731: Ok
- US006694305: Ok
- US006665678: Ok
- US006643637: Ok
- US006598041: Ok

OK | Back to List

Refine Search

Search Results -

Term	Documents
REUSE	97471
REUSES	2500
(11 AND REUSE).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	1
(L11 AND REUSE).PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD.	1

Database:

US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search: L12

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Monday, May 31, 2004 Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L12	l11 and reuse	1	L12
L11	L10 and (match$ or similar$)	13	L11
L10	L9 and (database near statement)	14	L10
L9	(database near cursor$) and l6	73	L9
L8	L6 and (similar near database near statement)	1	L8
L7	L6 and ((find$ or search$) near (similar near statement))	0	L7
L6	707/$.ccls.	20635	L6
	DB=USPT; PLUR=YES; OP=OR		
L5	L4 and reus$	8	L5
L4	L3 and 707/$.ccls.	40	L4

L3	similar near statement	511	L3
L2	(search$ or retriev$) near (similar near statement)	0	L2
L1	(search$ or retriev$) near (similar near statement) same database	0	L1

END OF SEARCH HISTORY

First Hit Fwd Refs

Generate Collection Print

L5: Entry 4 of 8 File: USPT Jul 31, 2001

DOCUMENT-IDENTIFIER: US 6269373 B1
TITLE: Method and system for persisting beans as container-managed fields

Brief Summary Text (8):
The Enterprise JavaBeans (EJB) component architecture is designed to enable enterprises to build scalable, secure, multi-platform, business-critical applications as reusable, server-side components. Its purpose is to solve the enterprise problems by allowing the enterprise developer to focus only on writing business logic.

Detailed Description Text (52):
With reference now to FIGS. 9A-9D, examples of Java programming language statements are shown that describe a distributed application in which a Java client invokes a remote business method from an Enterprise JavaBean running in a CORBA server. In FIGS. 9A, an example shows the standard RMI technique in a Java program. A remote object of the CustomerImpl class may have a set of business methods represented by an interface CustomerInterface. Methods included within CustomerInterface can be invoked remotely from a Java client. In this case, the client code would appear similar to statements 902-906. Statement 902 shows the use of a naming service to obtain an object reference for a remote object. Object "obj" is an RMI proxy of the remote Java object that implements CustomerInterface. After obtaining an object reference in statement 902, the object is narrowed by typecasting it to the appropriate object type in statement 904. In statement 906, the Java client code calls a business method on the proxy object as if it were a local object. The client is unaware that the call is implemented using an ORB. The proxy object transfers the method call to the remote object in a manner defined by the CORBA standard.

Current US Original Classification (1):
707/10

First Hit Fwd Refs

Generate Collection Print

L5: Entry 2 of 8 File: USPT Sep 24, 2002

DOCUMENT-IDENTIFIER: US 6457007 B1
TITLE: Distributed database management system including logical database constituted by a group of physical databases

Brief Summary Text (15):
However, in the distributed database environment where each database management system is operated independently, the accounts registered in the database management systems are not always the same. There is a method available for reregistering the accounts of all the database management systems and constructing a unified database account environment. However, for the reason of an increase in the account reregistration operation and reusing to the existing database application program property, many enterprises use a different account environment for each database management system continuously. Therefore, it is necessary for the user to selectively use the corresponding account for each database management system to be connected. However, in a large scale distributed database environment, in the same way as with the table location, storing and using of accounts to be used by the user for many database management systems require a very complicated operation.

Detailed Description Text (306):
As mentioned above, according to the logical database access function in this Embodiment 11, by a SELECT statement which is similar to a join for tables stored in a single physical database 1, a plurality of tables stored in different physical databases 1 in the same logical database can be joined.

Current US Original Classification (1):
707/10

Current US Cross Reference Classification (1):
707/100

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L10: Entry 5 of 5 File: USPT Jun 2, 1998

DOCUMENT-IDENTIFIER: US 5761654 A

**** See image for Certificate of Correction ****

TITLE: Memory structure and method for tuning a database statement using a join-tree data structure representation, including selectivity factors, of a master table and detail table

Abstract Text (1):
A method of constructing an aid to tuning of database statements comprises a data structure (stored in a memory on a computer system) which compactly represents that information which is needed about a database statement to determine the optimal series of operations (e.g., table data fetches) needed to execute the statement. The information includes the relationships between tables joined in the statement and the selectivities of logical conditions applied to rows in those tables.

Brief Summary Text (19):
Typically a user, either human or machine, sends an SQL statement to a database management program with SQL search capabilities, such as Oracle7 (.TM.), distributed by the assignee of this application, and other programs well known to those of ordinary skill. Such a program is referred to for convenience as a "search program." The search program interprets the SQL statement and combs through the database in search of information satisfying the statement. In the hypothetical database above, for example, a user might want to query the database to create a list of customers serviced by a particular sales representative.

Detailed Description Text (14):
select sum(decode(sign(sal-100000), 1, 1, 0))/count(*) from emp or using similar statements in other SQL dialects.

Current US Original Classification (1):
707/2

Current US Cross Reference Classification (1):
707/101

Current US Cross Reference Classification (2):
707/3

Current US Cross Reference Classification (3):
707/4